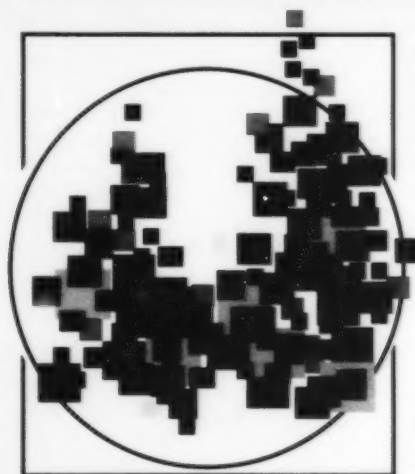
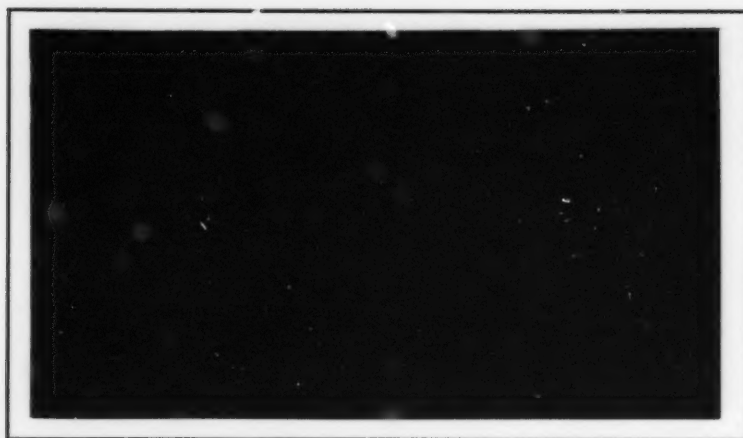




CENTRE FOR
SYSTEMS
SCIENCE

LCCR

**Tree Unification Grammar
Parsing and Inheritance Networks**

Fred Popowich

CSS/LCCR TR90-15
CMPT TR90-07

August 1990

Tree Unification Grammar, Parsing and Inheritance Networks

Fred Popowich

CMPT TR 90-07
CSS/LCCR TR 90-15

August 1990

School of Computing Science
Centre for Systems Science

Simon Fraser University
Burnaby, B.C.
CANADA V5A 1S6

Abstract

Parsing/generation of a unification-based grammar and inheritance in a semantic network are closely related. To show this, we introduce a restricted form of the Tree Unification Grammar formalism and develop a hierarchy of the the different concepts and structures used in parsing/generation with this formalism. This hierarchy is then expanded into an inheritance network which has a straightforward implementation in Prolog. The Prolog implementation of the network is then incorporated into a very simple parser/generator.

1. Introduction

Tree Unification Grammar (TUG) (Popowich 1988, 1989a) evolved from Unification Categorical Grammar (UCG) (Calder, Klein, and Zeevat, 1988) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard, 1985; Pollard and Sag, 1987). The TUG formalism uses partial descriptions of binary trees as its basic structures which are combined according to its single grammar rule to yield more fully specified descriptions of binary trees. The nodes in these trees are *signs* — complex feature structures containing information describing the morphological, syntactic and semantic properties of linguistic constituents.

In this paper, we will not be concerned with the use of TUG as a formalism for describing linguistic phenomena; instead, we shall examine some nonlinguistic properties of the formalism, and see how they relate to concepts studied in semantic network research. The focus will be on the relationship of TUG parsing to inheritance in a semantic network.

With this aim in mind, we shall first introduce a simplified version of TUG called *Restricted Tree Unification Grammar* (RTUG) in which simple grammar symbols (rather than complex signs containing linguistic information) are associated with the tree nodes. Then, we shall consider how the different concepts and structures associated with RTUG parsing can be related within an inheritance network. Finally, we shall give a characterization of the RTUG network in first order logic that can be executed in Prolog.

2. Restricted Tree Unification Grammar

RTUG, like TUG, possesses a single grammar rule which manipulates *functor argument (FA) specifications*. Before we discuss FA specifications, and how they are associated with the words and phrases of a language during parsing, we need to introduce some primitive structures and concepts.

A *functor argument (FA) tree* τ is a binary tree in which the nodes are labelled with simple grammar symbols. Each nonterminal node of an FA tree has a designated *functor* and *argument* subtree. Each node at the fringe of an FA tree is designated to be either a *terminal* or *pseudoterminal*. Intuitively, a pseudoterminal node corresponds to a node whose terminal or nonterminal status is unknown (there is no information about whether it has functor and argument subtrees). Based on the type of its root node, an FA tree will be either *terminal*, *nonterminal* or *pseudoterminal*.

The unification of FA trees can be defined recursively according to the different types of FA trees. Let τ_1 and τ_2 be FA trees, let *func* and *arg* be functions which return the functor and argument subtrees of a nonterminal FA tree, let *root* be a function for returning the root node of an FA tree, and let $\tau_1 \sqcup \tau_2$ be the FA tree resulting from the tree unification of τ_1 and τ_2 .

- If τ_1 and τ_2 are terminal trees and if $\text{root}(\tau_1) = \text{root}(\tau_2)$ then $\tau_1 \sqcup \tau_2 = \tau_1 = \tau_2$.
- If τ_1 and τ_2 are nonterminal trees and if $\text{root}(\tau_1) = \text{root}(\tau_2)$ then $\tau_1 \sqcup \tau_2 = \tau$ where $\text{func}(\tau), \text{arg}(\tau) \neq \perp$, $\text{root}(\tau) = \text{root}(\tau_1) = \text{root}(\tau_2)$, $\text{func}(\tau) = \text{func}(\tau_1) \sqcup \text{func}(\tau_2)$ and $\text{arg}(\tau) = \text{arg}(\tau_1) \sqcup \text{arg}(\tau_2)$.
- If τ_1 is a pseudoterminal tree and if $\text{root}(\tau_1) = \text{root}(\tau_2)$ then $\tau_1 \sqcup \tau_2 = \tau_2 \sqcup \tau_1 = \tau_2$, where τ_2 is any FA tree.

- Otherwise $\tau_1 \sqcup \tau_2 = \perp$ (i.e., unification fails).

An FA specification is a pair $\langle \tau, \alpha \rangle$ where τ is an FA tree and α is an auxiliary list. The auxiliary list is a (possible empty) list of subtrees of the FA tree.

Our formal definition of RTUG closely mirrors the one for lexicalized tree adjoining grammars (LTAGs) (Schabes, Abeille and Joshi, 1988, p.582). An RTUG is a quintuple $(V_N, V_T, \Theta, Lex, \theta_S)$ where:

- V_N is a finite set of nonterminal symbols,
- V_T is a finite set of terminal symbols,
- Θ is the set of FA specifications that can be constructed from FA trees whose nodes are labelled with elements of V_N and V_T ,
- Lex is the lexicon — a mapping from elements of V_T^* to finite subsets of Θ ,
- θ_S is an FA specification corresponding to the start symbol of the grammar.

Lex is a mapping from V_T^* not from V_T so, like with LTAG, we can have lexical entries for not just words but for phrases and empty sequences as well. A set of grammar rules is not a parameter of an RTUG — the single grammar rule is fixed.¹

The set of FA specifications can be divided into two disjoint sets based on whether they have empty or nonempty auxiliary lists. Those with empty auxiliary lists are *saturated*, the others are *unsaturated*. The single grammar rule combines an unsaturated FA specification (the *primary*) with a saturated FA specification (the *auxiliary*) to produce a new FA specification (the *result*). It effectively lays the auxiliary tree over the primary tree. This rule requires the first element of the auxiliary list of the primary to unify with the FA tree of the auxiliary to produce the FA tree of the result. The auxiliary list of the result is that of the primary less its first element. Adopting the convention that lists are enclosed in square brackets, and letting α be a variable over a sequence of list elements, the grammar rule can be characterized as follows:

$$\langle \tau_P, [\alpha] \rangle \rightarrow \langle \tau_P, [\tau_A, \alpha] \rangle, \langle \tau_A, [] \rangle$$

The language associated with a grammar can be defined in terms of a *sequent* (c.f. van Benthem, 1988). A sequent is of the form: $\theta_1, \theta_2, \dots, \theta_n \Rightarrow \theta$ where $\theta_i, \theta \in \Theta$, and θ is saturated. A sequent is reduced by an inference rule corresponding to the TUG grammar rule (c.f. Schabes, Abeille and Joshi, 1988). The grammar rule combines two FA specifications from the left hand side of the arrow in the sequent and replaces them with a new FA specification. This operation closely corresponds to function application in categorial grammar and UCG.

A sequent is said to be derivable (i.e., $\vdash \theta_1, \theta_2, \dots, \theta_n \Rightarrow \theta$) if it can be reduced by grammar rule applications to the sequent $\theta_F \Rightarrow \theta$ where θ_F is grounded (i.e., it does not contain any pseudoterminal nodes) and saturated, and where the FA trees of θ and θ' unify. We will refer to θ_F as the *FA structure* of the derivable sequent.

¹An alternative definition in which an RTUG is a sextuple including a set of production rules could be proposed.

The string language of an RTUG is defined to be:

$$\{a_1 a_2 \dots a_n \mid \exists \theta_i \in \text{Lex}(a_i), 1 \leq i \leq n, a_i \in V_T^* \text{ s.t. } a_1 a_2 \dots a_n = \text{fringe}(\theta), \mid - \theta_1, \theta_2, \dots, \theta_n \Rightarrow \theta_S\}$$

where θ is the FA structure associated with the derivable sequent.

The parsing process can then be viewed in terms of deduction (van Benthem, 1988; Schabes, Abeille and Joshi, 1988; König, 1989). Given a sentence to be parsed, we construct a sequent from the lexical entries associated with the words in the sentence. If this sequent is derivable, then we have a grammatical sentence.

As an example, consider the following context-free grammar for $\{a^n b^n \mid n \geq 1\}^2$ which is equivalent to the RTUG shown in Figure 2-1.

$S \rightarrow a, S, B$
 $S \rightarrow a, B$
 $B \rightarrow b$

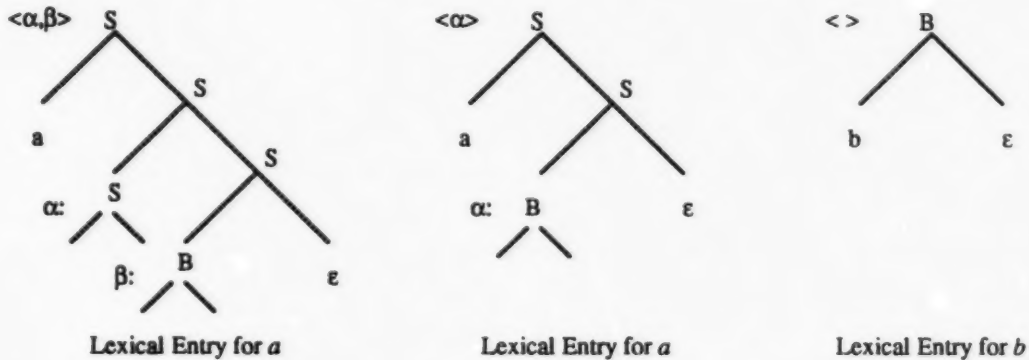


Figure 2-1: RTUG for $a^n b^n$

As an abbreviatory convention, the FA specifications in our diagrams will contain subtree labels rather than the actual subtrees themselves. Subtrees are labelled by Greek letters which precede the root node of the subtree.

Let us consider the derivation of the string "a a b b" with this grammar. In a successful parse of this sentence, one "a" from the sentence will require the first lexical entry for "a", with the other requiring the second lexical entry. Each occurrence of "b" will have the lexical entry for "b" associated with it. So in our parse of "a a b b", " $\text{Lex}(a) \text{Lex}(a) \text{Lex}(b) \text{Lex}(b)$ " will be the sequence shown in Figure 2-2. For the purpose of discussion we will distinguish between the two identical FA specifications for $\text{Lex}(b)$ — they are named θ_3 and θ'_3 . The second and third FA specifications θ_2 and θ_3 can be combined to give the sequence containing θ'_2 shown in Figure 2-3. Note that the auxiliary list of θ'_2 is obtained by removing α

²There is a straightforward mapping from context-free grammar in Greibach Normal Form (Greibach, 1965) to RTUG which is shown in Appendix B.

from the auxiliary list of θ_2 . The tree in θ'_2 is obtained by overlaying θ_3 at α in θ_2 . Then θ_1 and θ'_2 can be combined to yield θ'_1 . Finally, θ'_1 and θ'_3 can be combined to yield the saturated FA specification θ_F shown in Figure 2-4. The derivation of the final FA specification corresponds to the following derivation of sequents, where θ_S is the FA specification $\langle \tau_S, [] \rangle$ with τ_S being a pseudoterminal FA tree with root node S.

$$\theta_1, \theta_2, \theta_3, \theta_3 \Rightarrow \theta_S \quad | - \quad \theta_1, \theta'_2, \theta_3 \Rightarrow \theta_S \quad | - \quad \theta'_1, \theta_3 \Rightarrow \theta_S \quad | - \quad \theta_F \Rightarrow \theta_S$$

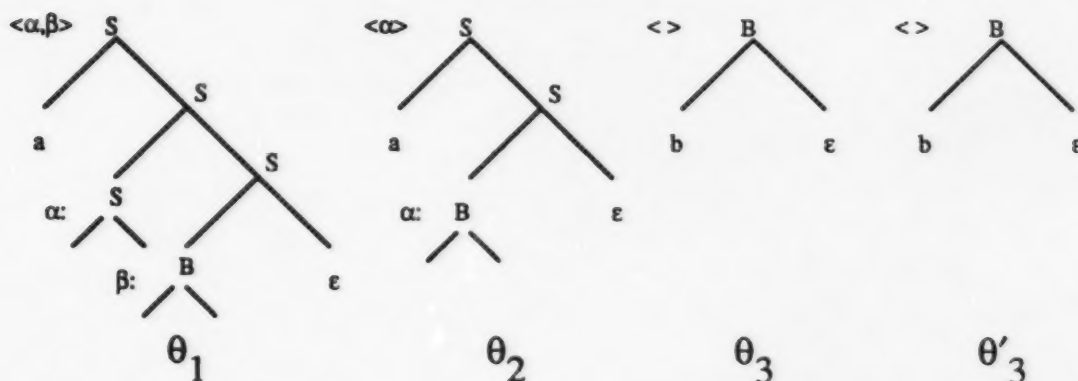


Figure 2-2: Initial Sequence of FA Specifications

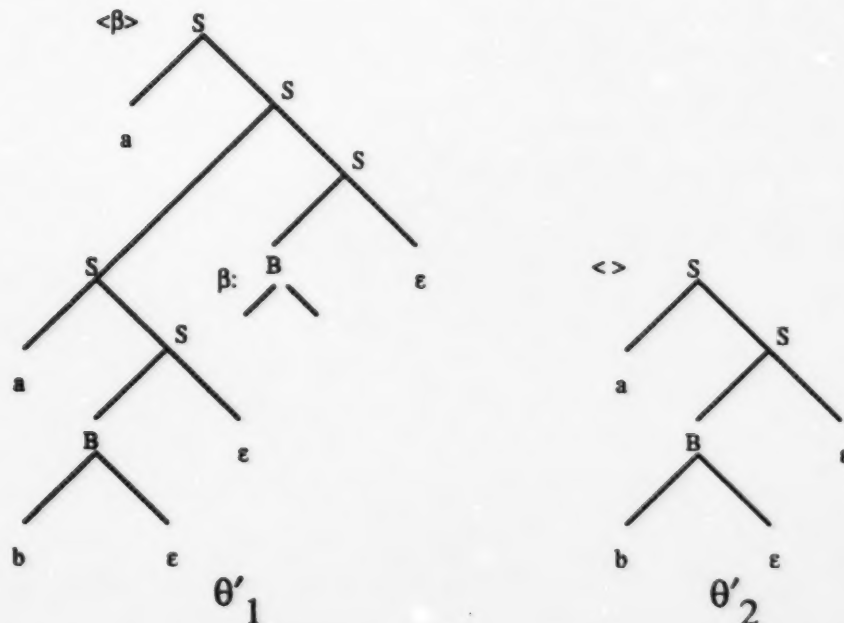


Figure 2-3: Derived FA Specifications

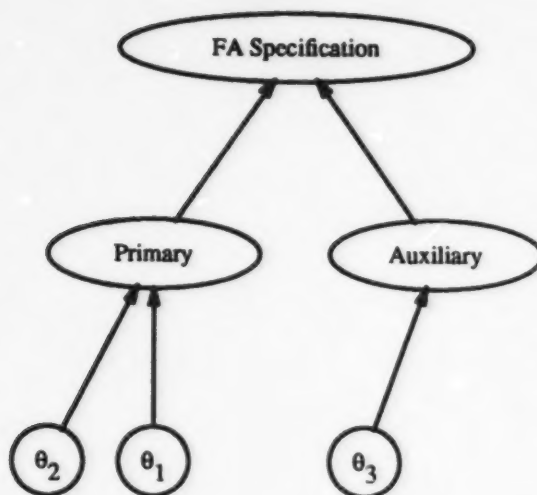


Figure 3-1: FA Specification Hierarchy

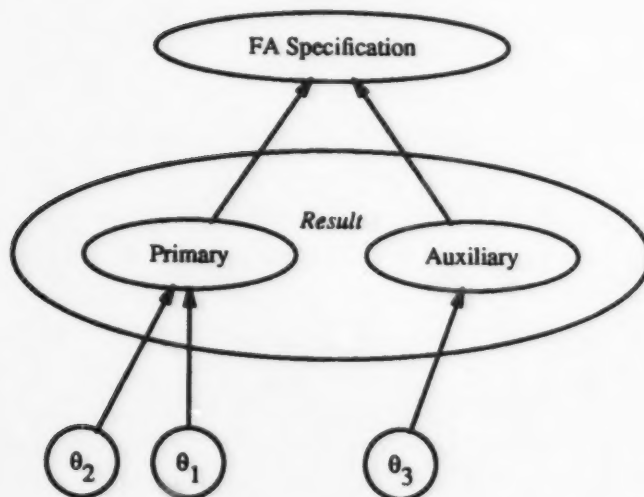


Figure 3-2: Inclusion of Grammar Rule in Hierarchy

Also note that although θ_F was not used as an auxiliary in the invocation of a grammar rule, this same tree does have the potential to act as an auxiliary in other derivations according to the grammar (e.g., in a derivation for the sentence "a a a b b b").

So far, we have only been looking at a network associated with a particular set of lexical entries and for a particular parse. If we introduce concept nodes for lexical entries (like θ_1 , θ_2 and θ_3) and derived FA specifications (like θ'_1 , θ'_2 and θ_F), then we obtain the network shown in Figure 3-4.

Note that we are no longer dealing with specific instances of derived FA specifications or of lexical

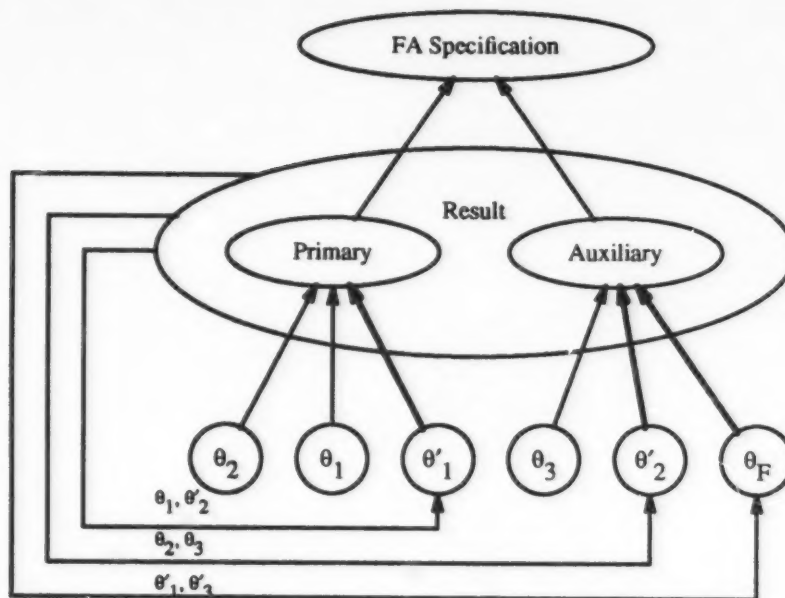


Figure 3-3: Network for Sample Grammar

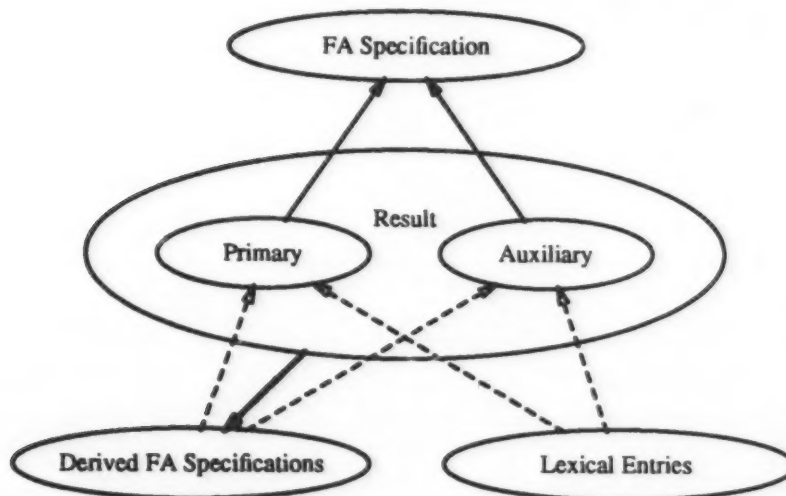


Figure 3-4: RTUG Network

entries; we are dealing with the class of results not with individual results. The two dashed arcs leading from the *derived* node and the two leading from the *lexical* node are defeasible links. They capture the relationship that a lexical entry is either an auxiliary or a primary, and that a derived FA specification is similarly either an auxiliary or a primary.

Adopting the notation used by Hendrix (1975, 1979), the subnet of Figure 3-4 containing only the

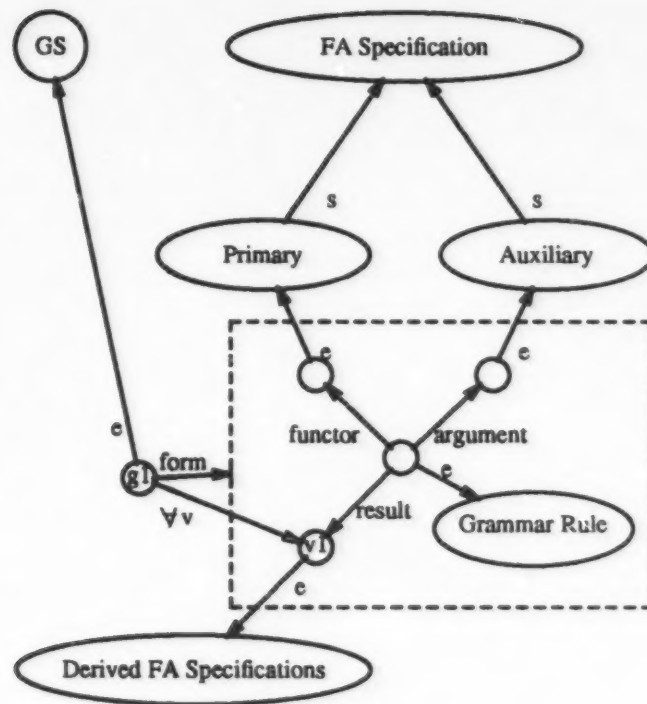


Figure 3-5: Partitioned Semantic Network

solid links could be expressed as shown in Figure 3-5. In this figure, arcs are labelled with "e" are used to describe "isa" relationships between instances and classes, while arcs labelled with "s" represent "subtype" relations between classes. The *g1* node, which is an instance of a *General Statement* (GS), describes the set of all objects *v1* that satisfy relationship contained within the supernode accessed by "form". Essentially, the network is interpreted as meaning that anything that is the result of a grammar rule application "isa" *Derived FA Specification*, where the grammar rule application requires a functor which "isa" *Primary*, and an argument which "isa" *Auxiliary* FA specification. This network could be extended to incorporate the information from the dashed arcs in Figure 3-4 by introducing additional subnets and variable nodes, along with nodes for the predicates *saturated* and *unsaturated*.

We can actually give the semantics of the network introduced in Figure 3-4 in terms of first order predicate calculus, as will be illustrated by a straightforward Prolog implementation of this network in the next section.

4. Implementing the Hierarchies

Actual instances of FA specifications are implemented as Prolog terms which pair together representations of trees and corresponding auxiliary lists. The colon separates the FA tree from the auxiliary list in the Prolog representation. Our term representation for FA trees will be based on a standard representation for trees with atomic nodes (Clocksin and Mellish, 1981). Any binary tree with

root node n which has the subtrees α and β as its functor and argument respectively will be represented by the term $n(\alpha', \beta')$, where α' and β' are the term representations of α and β respectively. Depth zero trees rooted by a terminal node t will be represented by $t(\text{nil}, \text{nil})$, while those rooted by a pseudoterminal node p will be represented by $p(U, V)$ where U and V are distinct variables associated with the pseudoterminal.³ Auxiliary lists are implemented as standard Prolog lists: list elements are enclosed in square brackets and separated by commas. The notation "[*Head/Rest*]" is used for a list whose first element is *Head* where *Rest* is a list of the remaining elements.

Consider the term representations of the lexical entries first introduced in Figure 2-1.

```
'FA' (s(a(nil, nil), s(s(U1, U2), s(b(U3, U4), e(nil, nil))))):
    [b(U3, U4), s(U1, U2)]).
'FA' (s(a(nil, nil), s(b(U1, U2), e(nil, nil)))):
    [b(U1, U2)]).
'FA' (b(b(nil, nil), e(nil, nil))):
    []).
```

In the above representations of the lexical entries, each element of the auxiliary list is some subterm (and thus a subtree) of the term corresponding to the tree. Recall from §2 that a pseudoterminal node will unify with either a terminal or a nonterminal node, but a terminal node will not unify with a nonterminal.

The hierarchy of FA specifications from Figure 3-1 is implemented by the following Prolog clauses.

```
fa(FA)      :- auxiliary(FA).           % auxiliary SUBTYPE fa
fa(FA)      :- primary(FA).             % primary SUBTYPE fa
primary(FA)  :- unsaturated(FA), 'FA'(FA). % unsaturated-FA ISA primary
auxiliary(FA) :- saturated(FA), 'FA'(FA). % saturated-FA ISA auxiliary
```

Also recall that auxiliary FA specifications are distinguished from primary FA specifications based on whether or not they are saturated (i.e., whether they have an empty or a nonempty auxiliary list). This distinction is determined by the *saturated* and *unsaturated* predicates whose definitions are shown below.

```
unsaturated(_: [_|_]).
saturated(_: []).
```

The grammar rule of TUG is responsible for establishing a relationship between a primary FA specification, an auxiliary FA specification and some resulting FA specification (as was illustrated graphically in Figure 3-2). This relationship can be expressed as follows.

```
'FA'(Res) :- auxiliary(Aux), primary(Pri), grammar_rule(Pri, Aux, Res).
grammar_rule(P:[A|Rest], A:[], P:Rest).
```

It is the definition of *grammar_rule* that contains the different aspects of the RTUG grammar rule and determines which information is *inherited* to the resulting FA specification. It requires:

- the auxiliary FA tree, *A*, to be the same as the first element of the auxiliary list of the primary;
- the auxiliary list of the result, *Rest*, to be the same that of the primary less its first element;
- the auxiliary list of the auxiliary to be empty; and
- the FA tree of the result to be the same as that of the primary.

³Variables start with either uppercase or underscores.

Note that *P* will be modified by application of the grammar rule since *A* will be some subtree of *P*.

By incorporating the restrictions contained in the *saturated*, *unsaturated* and *grammar_rule* predicates directly into the definitions of the semantic network classes and objects, and introducing two more classes of FA specifications corresponding to those introduced in Figure 3-4, the result is the final Prolog representation shown below in which the semicolon denotes disjunction.

```
fa(FA)           :- auxiliary(FA) ; primary(FA).
primary(T:[A|B]) :- lexical(T:[A|B]) ; derived(T:[A|B]).
auxiliary(T:[])  :- lexical(T:[]) ; derived(T:[]).

derived(P:Rest)  :- auxiliary(A:[]), primary(P:[A|Rest]).
```

Note that the traversal of any of the arcs to primary, auxiliary or derived is associated with the inheritance of specific information about the structure of the FA specification. For example, the traversal of the arc from *Lexical Entries* to *Auxiliary* has associated with it the information that the object in question (i.e., the argument of the *lexical* predicate) be saturated (i.e., have an empty auxiliary list). As discussed in (Vogel and Popowich 1990; Vogel forthcoming), if the object associated with the traversal of this arc has conflicting information then the arc is interpreted as a negative link, otherwise the information about being saturated is inherited through this link.

Based on the definition of the string language associated with an RTUG, the following definition of the sentence predicate can be given.

```
sentence(S) :- start_FA(FA), auxiliary(FA), grounded(FA), fringe(FA,S).

start_FA(s(_,_) : []).
```

This definition, which uses the above representation of the concept hierarchy, will work as both parser and generator. It is not designed for efficiency; rather, it is designed to be concise and descriptive. As a parser, it works in a generate and test manner, looking for an auxiliary (saturated) FA specification that unifies with the start symbol of the grammar. When such an FA specification from the concept hierarchy is found, it is checked to see if it is grounded and whether its fringe matches the input sentence *S*. The efficiency of this search could easily be improved by considering only those lexical entries associated with the different elements of *S*. The definitions of the *grounded* and *fringe* predicates can be found in Appendix A.

Test runs for the final Prolog description, together with the lexical entries described above, are also included in Appendix A. Predicate calls that use the Prolog implementation of the RTUG network are executed with a depth-bounded search strategy. This use of a depth-bounded strategy allows the concise Prolog description to be used without the need for additional tests to eliminate infinite looping. It also allows us to generate all possible FA specifications and sentences that can be found with a bounded search. The implementation exhibits a large amount of duplication of effort whenever it is looking for derived FA specifications: a derived FA specification is recomputed whenever it is needed.

5. Conclusion

We have shown how the different classes of FA specifications associated with parsing and generation in RTUG can be organized into a semantic network containing object nodes, class nodes and partitions or supernodes. While most of the arcs in this network are either traditional "isa" or "subtype" links that are usually found in inheritance hierarchies, others have restrictions on them limiting their applicability or equivalently specifying information that is inherited when the link is traversed. This network is equivalent to a first-order logic description that can be directly translated into Prolog. The Prolog specification of the network can be used both in the generation and parsing of FA specifications and of sentences in the string language of an RTUG. Although the parser/generator is not efficient, it does illustrate how the concept hierarchy can be used directly by a parser. Some simple modifications (like only considering lexical entries for those words appearing in the sentence being parsed) would make the implementation more efficient, and the inclusion of some additional tests could solve the termination problems that the parser exhibits. As it stands, however, the same description can be used for both generation and analysis. A chart parser for TUG that does not suffer from these efficiency and termination problems and which takes advantage of specific properties of TUG is described in (Popowich, 1989b).

The relationship between inheritance hierarchies and parsing outlined in this paper is not restricted to just RTUG. There is a similar relationship for HPSG (Vogel and Popowich, 1990; Vogel, forthcoming) which could be extended to other unification-based (also known as information based) grammar formalisms. What links these formalisms to inheritance hierarchies is that both inheritance and unification deal with processes for the addition of compatible information.

Acknowledgements

I would like to extend special thanks to Carl Vogel for his original insight into the relationship between parsing and inheritance networks, and for numerous discussions with him about earlier drafts of this paper, and about inheritance in general. I am grateful to Dan Fass and to Veronica Dahl for their comments on drafts of this paper.

This research is supported by a research fellowship from the Advanced Systems Institute of British Columbia and by the Natural Sciences and Engineering Research Council of Canada under Grant No. OGP0041910.

References

- Brachman, Ronald J. (1983). What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. *IEEE Computer*, 16(10), 30-38.
- Calder, Jo, Ewan Klein, and Henk Zeevat. (1988). Unification Categorical Grammar: A Concise, Extendable Grammar for Natural Language Processing. In: *12th International Conference on Computational Linguistics*. Budapest, Hungary.
- Clocksin, William, and Chris Mellish. (1981). *Programming in Prolog*. Springer-Verlag, Berlin, Heidelberg, West Germany.

Greibach, S. (1985). A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *Journal of the ACM*, 12(1), 42-52.

Hendrix, Gary. (1975). Expanding the Utility of Semantic Networks Through Partitioning. In: *Proceedings of the 4th International Joint Conference on Artificial Intelligence*. Tbilisi, U.S.S.R..

Hendrix, Gary. (1979). Encoding Knowledge in Partitioned Networks. In N. Findler (Ed.), *Associative Networks: The Representation and Use of Knowledge by Machine*. Academic Press, New York.

König, Ester. (1989). Parsing as Natural Deduction. In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, B.C.

Pollard, Carl. (1985). Lectures on HPSG. Unpublished lecture notes, CSLI, Stanford University, CA.

Pollard, Carl, and Ivan Sag. (1987). *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. Centre for the Study of Language and Information, Stanford University, CA.

Popowich, Fred. (1988). *Reflexives and Tree Unification Grammar*. Doctoral dissertation, Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland.

Popowich, Fred. (1989a). Tree Unification Grammar. In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, B.C.

Popowich, Fred. (1989b). *A Tree Unification Grammar-Based Natural Language Processor* (CSS-IS TR 89-06). Simon Fraser University, Burnaby, B.C.

Schabes, Yves, Anne Abelle, and Aravind Joshi. (1988). Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In: *12th International Conference on Computational Linguistics*. Budapest, Hungary.

van Benthem, Johan. (1988). The Lambek Calculus. In R.T. Oehrie, E. Bach and D. Wheeler (Eds.), *Categorical Grammars and Natural Language Structures*. D. Reidel, Dordrecht, Holland.

Vogel, Carl. (forthcoming). *Two Extensions to Inheritance Reasoning (working title)*. Master's thesis, Simon Fraser University, Burnaby, B.C.,

Vogel, Carl and Fred Popowich. (1990). Head-Driven Phrase Structure Grammar as an Inheritance Hierarchy. In W. Daelemans and G. Gazdar (Eds.), *Inheritance in Natural Language Processing Workshop Proceedings*. Institute for Language Technology and Artificial Intelligence, Tilburg University, Holland. also appeared as CSS/LCCR TR 90-03, Centre for Systems Science, Simon Fraser University, Burnaby, B.C.

A. Terminal Session

```

let cat interp.pl
% interp.pl          Bounded Depth-First Execution

:- op(900, xfx, '?').

Depth ? Goal      :- length(D,Depth), bprove(Goal,D).

bprove( true, _ ) :- !.
bprove( (A,B), D ) :- !, bprove(A,D), bprove(B,D).
bprove( (A;_), D ) :- bprove(A,D).
bprove( (_,B), D ) :- !, bprove(B,D).
bprove( Goal, [_|D] ) :- clause(Goal,Body), bprove(Body,D).

let cat tugnet.pl
:- dynamic fa/1, primary/1, auxiliary/1, derived/1, lex/1.
:- op(600, xfx, ':').

lex(b(b(nil,nil),e(nil,nil)):[]).
lex(s(a(nil,nil),s(b(U1,U2),e(nil,nil)))):[b(U1,U2)]).
lex(s(a(nil,nil),s(a(U1,U2),s(b(U3,U4),e(nil,nil))))):[b(U3,U4),s(U1,U2)]).

derived(P:Rest) :- auxiliary(A:[]), primary(P:[A|Rest]).

primary(T:[A|B]) :- lex(T:[A|B]); derived(T:[A|B]).

auxiliary(T:[]) :- lex(T:[]); derived(T:[]).

fa(FA)          :- auxiliary(FA); primary(FA).

let cat parser.pl
% Use depth-bounded execution of "auxiliary" (depth=16)

sentence(S) :- start_FA(FA), !, auxiliary(FA), grounded(FA), fringe(FA,S).

start_FA(s(_,_):[]) :- !, % Start symbol of Grammar

grounded(Tree:_) :- !, grounded(Tree). % For FA Specifications

grounded(V) :- var(V), !, fail.
grounded(Tree) :- Tree == [_|Func,Arg], !, grounded(Func), grounded(Arg).
grounded(_).

fringe(Tree:_, Fringe) :- !, fringe(Tree, Fringe).

fringe(Tree, []) :- Tree == [e,nil,nil], !. % For empty symbols
fringe(Tree, [Root]) :- Tree == [Root,nil,nil], !. % For terminal symbols
fringe(Tree, Fringe) :- Tree == [_|Func,Arg], fringe(Func, F), fringe(Arg, A), append(F, A, Fringe).

append([],L,L).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).

let cat prolog

Quintus Prolog Release 2.4.2 (Sun-4, SunOS 4.0)
Copyright (C) 1988, Quintus Computer Systems, Inc. All rights reserved.
1310 Villa Street, Mountain View, California 415) 965-7700

| ?- compile(interp), consult(tugnet), compile(parser).
[compiling /home/dk3/popowich/fred/Research/Implementations/RTUG/interp.pl...]
[interp.pl compiled 0.166 sec 736 bytes]
[consulting /home/dk3/popowich/fred/Research/Implementations/RTUG/tugnet.pl...]
[tugnet.pl consulted 0.133 sec 1,344 bytes]
[compiling /home/dk3/popowich/fred/Research/Implementations/RTUG/parser.pl...]
[parser.pl compiled 0.300 sec 1,028 bytes]

yes
| ?- 107b(X).

X = b(b(nil,nil),e(nil,nil)):[] ;

X = s(a(nil,nil),s(b(b(nil,nil),e(nil,nil)),e(nil,nil)):[] ;

X = s(a(nil,nil),s(s(a(nil,nil),s(b(b(nil,nil),e(nil,nil))),s(b(b(nil,nil),e(nil,nil)),e(
nil,nil)))):[] ;

X = s(a(nil,nil),s(s(a(nil,nil),s(s(a(nil,nil),s(b(b(nil,nil),e(nil,nil))),s(b(b(nil,nil),
e(nil,nil)),e(nil,nil))))),s(b(b(nil,nil),e(nil,nil)),e(nil,nil)):[] ;

```

```

X = s(a(nil,nil),s(b(_243,_244),s(nil,nil))):[b(_243,_244)] ;
X = s(a(nil,nil),s(s(_243,_244),s(b(_249,_250),s(nil,nil)))):[b(_249,_250),s(_243,_244)] ;
X = s(a(nil,nil),s(s(_394,_395),s(b(b(nil,nil),s(nil,nil)),s(nil,nil)))):[s(_394,_395)] ;
no
| ?- sentence(S).
S = [a,b] ;
S = [a,a,b,b] ;
S = [a,a,a,b,b,b] ;
S = [a,a,a,a,b,b,b,b] ;
S = [a,a,a,a,a,b,b,b,b,b] ;
S = [a,a,a,a,a,a,b,b,b,b,b,b] ;
S = [a,a,a,a,a,a,a,b,b,b,b,b,b,b] ;
S = [a,a,a,a,a,a,a,a,b,b,b,b,b,b,b,b] ;
no
| ?- halt.

```

B. Context Free Grammars

For any context-free grammar in Greibach Normal Form (GNF), we can obtain an equivalent RTUG. Context-free grammars in GNF have rules of the form:

$$A \rightarrow a, B_1, \dots, B_n$$

where $A, B_i \in V_N$ with $n \geq 0$, $i \leq n$, and $a \in V_T$. The lexical entry for "a" associated with this rule is the FA specification shown in Figure B-1. The symbols associated with the unlabelled nodes along the spine of the tree are not relevant in this example, we could simply use the same label as appears on the root node.

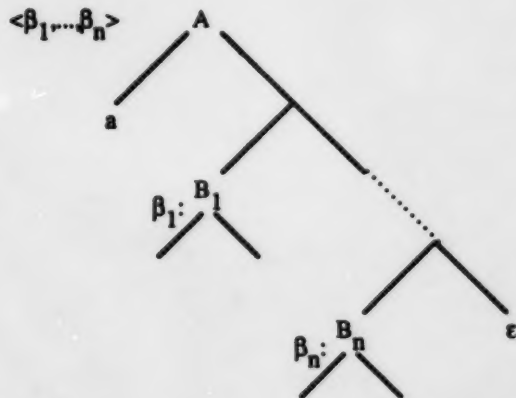


Figure B-1: FA Specification for Greibach Normal Form Grammar Rule

Centre for Systems Science

- Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.
- CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.
- CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.